



TITLE:

Listing All Trees with Specified Degree Sequence (Acceleration and Visualization of Computation for Enumeration Problems)

AUTHOR(S):

Nakano, Shin-ichi

CITATION:

Nakano, Shin-ichi. Listing All Trees with Specified Degree Sequence (Acceleration and Visualization of Computation for Enumeration Problems). 数理解析研究所講究録 2009, 1644: 55-62

ISSUE DATE:

2009-04

URL:

<http://hdl.handle.net/2433/140650>

RIGHT:

Listing All Trees with Specified Degree Sequence

Shin-ichi Nakano

Department of Computer Science, Faculty of Engineering,
Gunma University, Kiryu 376-8515, Japan

Abstract

In this paper we designed a simple algorithm to generate all ordered trees with specified degree sequence. The algorithm generates each tree in $O(1)$ time for each on average.

1 Introduction

Generating all graphs having some property without duplicates has many applications, including unbiased statistical analysis[M98]. A lot of algorithms to solve these problems are already known, and can be found in good textbooks [G93, KS98, K06].

Trees are one of basic model frequently used in many areas, including searching for keys, modeling computation, parsing a program, etc.

Given a rooted tree T with n inner (non-leaf) vertices, the degree sequence of T is the list of n integers such that (1) each integer corresponds to the number of children of each inner vertex in T , and (2) the integers appear in nonincreasing order. Note that each rooted tree has a unique degree sequence, while a degree sequence may correspond to many rooted trees.

There are some algorithms to generate all ordered trees having specified degree sequence. The algorithm in [ZR79] generates all such ordered tree in $O(n)$ time for each, and loopless algorithms in [KL99, KL00, KL02] generate all such ordered trees in $O(1)$ time for each.

In this paper first we give a simple algorithm to generate all ordered trees having specified degree sequence in $O(1)$ time for each.

The outline of our algorithm is as follows.

Let O_D be the set of all ordered trees having specified degree sequence. First we define a tree structure FT_D among the trees in O_D so that each vertex in FT_D corresponds to each tree in O_D . Next we design a simple but efficient algorithm to compute all child vertices of a given vertex in FT_D . Applying the algorithm recursively from the root of FT_D , we can list all vertices in FT_D , and also corresponding

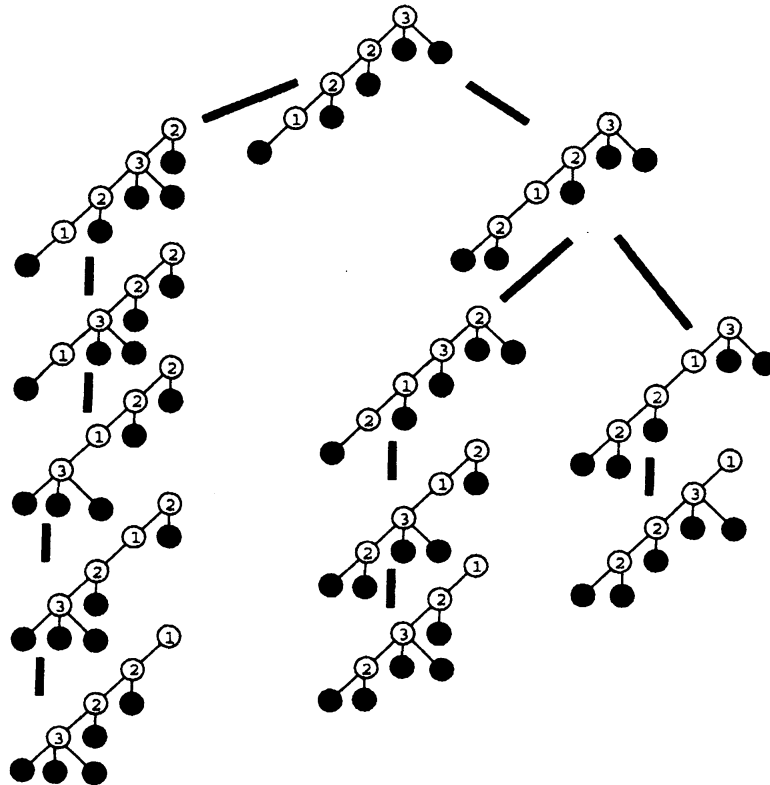


Figure 1: The family tree of O_D^1 .

trees in O_D . Many listing algorithms have designed based on such tree structures but with some other ideas[LN01, N02, N04, NU04].

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 defines the tree structure FT_D among O_D . Section 4 gives a simple but efficient algorithm to list all trees in O_D . Our algorithm generates all ordered trees with specified degree sequence in $O(1)$ time for each. Finally Section 5 is a conclusion.

2 Preliminary

A graph is a *tree* if it is connected and has no cycle. A tree T is *rooted* if one vertex r is designated as the *root* of T .

For each vertex v in a rooted tree, let $P(v)$ be the unique path from v to the root r . The *depth* of v is the number of edges in $P(v)$. The *parent* of $v \neq r$ is its neighbor on $P(v)$, and the *ancestors* of v are the vertices on $P(v)$. The parent of r is not defined. We say if v is the parent of u then u is a *child* of v , and if v is an ancestor of u then u is a *descendant* of v . Note that each vertex is always a descendant of itself. We denote by $d(v)$ the number of children of v . The *height* of a vertex v is the number of edges on the longest path from v to a descendant of v ,

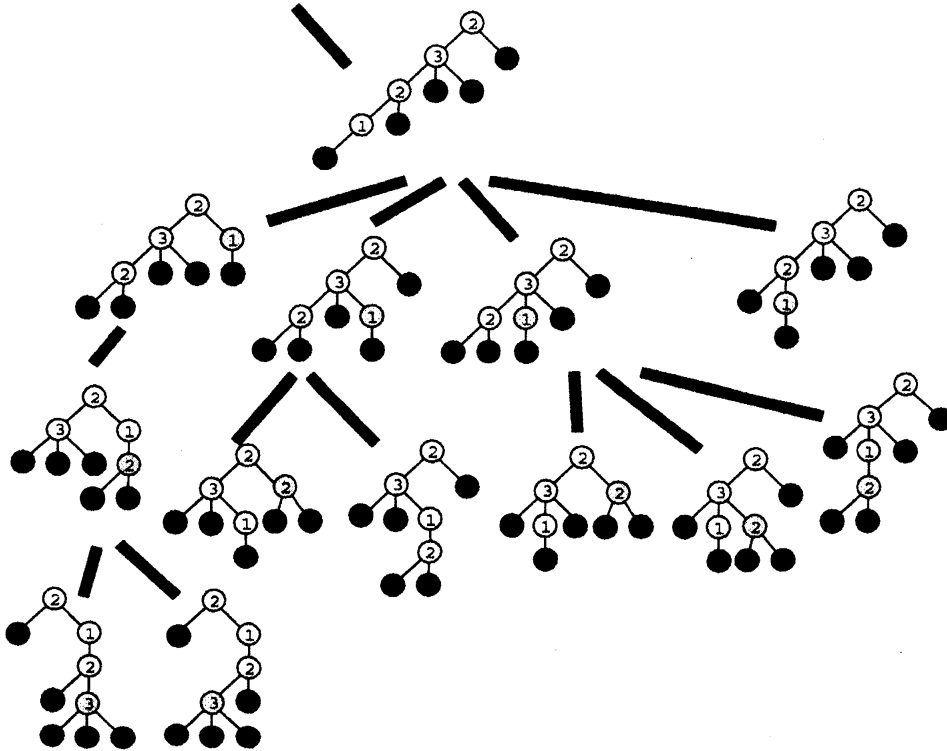


Figure 2: A subtree of the family tree FT_D .

and denoted by $h(v)$. A vertex is a *leaf* if it has no child, otherwise it is an *inner vertex*. The height of a leaf is always 0, and the height of a vertex is always larger than the height of its child by one.

The degree sequence of a rooted tree T having n inner vertices is the list of n integers such that (1) each integer corresponds to the number of children of each inner vertex in T , and (2) the integers appear in nonincreasing order. Note that each rooted tree has a unique degree sequence, while a degree sequence may correspond to many rooted trees.

Assume that $D = (d_1, d_2, \dots, d_n)$ is the degree sequence of a rooted tree T . Let n_i be the number of occurrences of integer i in D . Then the number of edges in T is $\sum_{i=1}^{n-1} i n_i$.

A rooted tree is an *ordered tree* if the children of each vertex are ordered linearly left-to-right, otherwise, it is an *unordered tree*.

3 The Family Tree

Let O_D be the set of all ordered trees having specified degree sequence $D = (d_1, d_2, \dots, d_n)$. In this section we define a tree structure FT_D among the trees in O_D . Then in the next section we give a simple but efficient algorithm to list all ordered trees in O_D .

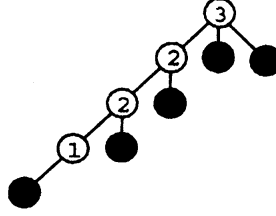


Figure 3: The root tree T_r^D of O_D where $D = (3, 2, 2, 1)$.

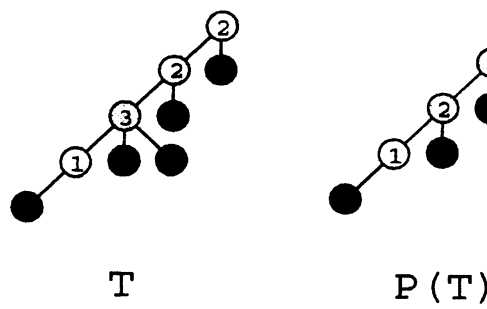


Figure 4: Illustration for Case 1.

Assume that T is an ordered tree.

The last inner vertex of T in preorder is called *the pruning vertex* of T . Note that all the child vertices of the pruning vertex are leaves.

The path $(\ell_1, \ell_2, \dots, \ell_q)$ in T is called *the left-down path* of T if (1) ℓ_1 is the root, (2) the leftmost child of ℓ_q is a leaf, and (3) ℓ_{i+1} is the leftmost child of ℓ_i for each $i = 1, 2, \dots, q - 1$. The leftmost child of ℓ_q is called *the leftmost leaf* of T .

Given $D = (d_1, d_2, \dots, d_n)$, let T_r^D be the ordered tree derived from the path $(\ell_1, \ell_2, \dots, \ell_n)$ by attaching $d_i - 1$ leaves to ℓ_i for $i = 1, 2, \dots, n - 1$ and d_n leaves to ℓ_n so that $(\ell_1, \ell_2, \dots, \ell_n)$ is the left-down path of T_r^D . See an example in Fig. 3. Thus $T_r^D \in O_D$ and $O_D \neq \emptyset$ holds. The ordered tree T_r^D is called *the root tree* of O_D .

For each ordered tree $T \in O_D - \{T_r^D\}$ with $D = (d_1, d_2, \dots, d_n)$, we define an ordered tree, called *the parent tree* $P(T)$ of T , as follows. We have two cases. Note that for each case T and $P(T)$ have the same degree sequence. Thus $P(T) \in O_D$ holds. Let $I(T)$ be the subgraph of T induced by all inner vertices of T .

Case 1: $I(T)$ is the left-down path of T .

Let $LD = (\ell_1, \ell_2, \dots, \ell_n)$ be the left-down path of T . Since $d(\ell_1) \geq d(\ell_2) \geq \dots \geq d(\ell_n)$ holds only for T_r^D , and by assumption $T \in O_D - \{T_r^D\}$, there is some i such that $d(\ell_i) < d(\ell_{i+1})$. Let a be the smallest index such that $d(\ell_a) < d(\ell_{a+1})$. $P(T)$ is the ordered tree derived from T by (1) removing $d(\ell_{a+1}) - d(\ell_a)$ child leaves from ℓ_{a+1} , then (2) attaching the removed child leaves to ℓ_a so that the left-down path

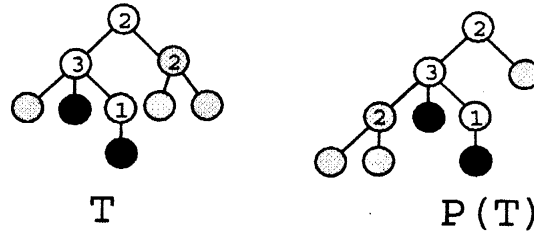
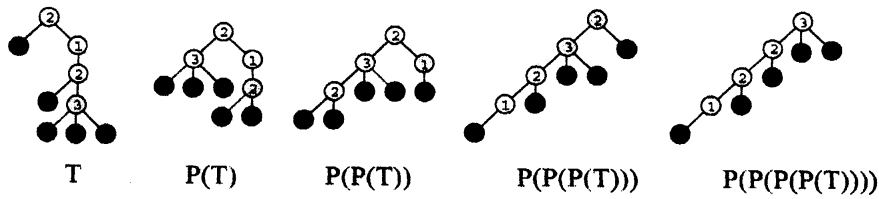


Figure 5: Illustration for Case 2.

Figure 6: The sequence $T, P(T), P(P(T)), \dots$.

remain as it was. See an example in Fig. 4. Intuitively $P(T)$ is derived from T by swapping ℓ_a and ℓ_{a+1} .

Case 2: $I(T)$ is not the left-down path of T .

$P(T)$ is the ordered tree derived from T by swapping (1) the subtree consisting of the pruning vertex p of T and its children, and (2) the leftmost leaf ℓ_q of T . See an example in Fig. 5. Note that all children of p are leaves since p is the last inner vertex in preorder. Also note that p is not in $I(T)$ since Case 1 does not occur.

Let O_D^1 be the subset of O_D consisting of all T such that $I(T)$ is the left-down path of T . If $I(T)$ is the left-down path of T then $P(T)$ is defined by Case 1 and $I(P(T))$ is also the left-down path of T . For any $T \in O_D^1 - \{T_r^D\}$, repeatedly finding the parent tree of the derived tree results in the sequence $T, P(T), P(P(T)), \dots$, which always end at the root tree T_r^D of O_D . By merging the sequence above for each $T \in O_D^1$ we can define a tree structure among trees in O_D^1 . See an example in Fig. 1.

Also note that if $I(T)$ is not the left-down path of T then $P(T)$ is defined by Case 2 and the number of vertices in the left-down path of $P(T)$ is increased by one from that of T . Again repeatedly finding the parent tree of the derived tree results in the sequence $T, P(T), P(P(T)), \dots$, in which Case 1 eventually occurs somewhere, and after that the sequence always end at the root tree T_r^D of O_D as mentioned above. See an example in Fig. 6.

By merging the sequence above for each $T \in O_D - \{T_r^D\}$ we can define the family tree FT_D , in which each vertex in FT_D corresponds to a tree in O_D , and each edge corresponds to each relation between some T and $P(T)$. See an example in Fig. 2.

4 Listing Ordered Trees

In this section we give a simple but efficient algorithm to list all ordered trees in O_D .

If we have an algorithm to list all child trees of an ordered tree in O_D , then by recursively applying the algorithm starting at the root tree T_r^D , we can list all ordered trees in O_D . Now we are going to design such an algorithm.

Let T be an ordered tree in O_D . We have two cases. Note that $T \in O_D^1$ means $I(T)$ is the left-down path of T .

Case 1: $T \in O_D^1$.

In this case T may have some child trees both in O_D^1 and $O_D - O_D^1$. Let $(\ell_1, \ell_2, \dots, \ell_n)$ be the left-down path of T . Since $I(T)$ is the left-down path of T all but the leftmost children of ℓ_i are leaves for each $i = 1, 2, \dots, n-1$, and all children of ℓ_n are leaves.

Child trees in O_D^1

Let $T[i]$ be the ordered tree derived from T by transferring some leaf children of either ℓ_i or ℓ_{i+1} to the other so that (1) the degree of ℓ_i and ℓ_{i+1} are exchanged and (2) the left-down path remains as it was.

By the definition of the parent tree in Section 3, each child tree T_c of T in O_D^1 is $T[i]$ for some i . However not all $T[i]$ are child trees of T . $T[i]$ is a child tree of T only if $P(T[i]) = T$ holds.

If $T = T_r^D$, then $d(\ell_1) \geq d(\ell_2) \geq \dots \geq d(\ell_n)$ holds, and $T[i]$ is a child tree of T for each $i = 1, 2, \dots, n-1$ if $d(\ell_i) < d(\ell_{i+1})$.

Otherwise, $d(\ell_1) \geq d(\ell_2) \geq \dots \geq d(\ell_n)$ does not hold. Let s be the smallest index such that $d(\ell_s) < d(\ell_{s+1})$. Now $T[i]$ is a child tree of T for each $i = 1, 2, \dots, s-1$ if $d(\ell_i) < d(\ell_{i+1})$. $T[s]$ is not a child of T . $T[s+1]$ is a child tree of T only if $d(\ell_{s+2}) \leq d(\ell_s)$. $T[i]$ is not a child tree of T for each $i = s+2, s+3, \dots, n-1$.

Note that if $T[i]$ is a child tree of T then the index s of $T[i]$ is always i .

Child trees in $O_D - O_D^1$

For each i, j such that $i = 1, 2, \dots, n-1$ and $j = 2, 3, \dots, d(\ell_i)$, let $T[i, j]$ be the ordered tree derived from T by swapping (1) the subtree rooted at ℓ_n and (2) the j -th child of ℓ_i . Note that all children of ℓ_n are leaves.

By the definition of the parent tree in Section 3, for each i, j such that $i = 1, 2, \dots, n-1$ and $j = 2, 3, \dots, d(\ell_i)$, $T[i, j]$ is a child tree of T .

Case 2: $T \notin O_D^1$.

In this case T has no child tree in O_D^1 , since the parent of each tree in O_D^1 is also in O_D^1 . However T may have child trees in $O_D - O_D^1$.

Let $(\ell_1, \ell_2, \dots, \ell_q)$ be the left-down path of T .

The path (r_1, r_2, \dots, r_p) is the *right-down path* of T if (1) r_1 is the root, (2) all child of r_p are leaves, and (3) r_{i+1} is the rightmost non-leaf child of r_i . Let (r_1, r_2, \dots, r_p) be the right-down path of T for each $i = 1, 2, \dots, p-1$. For $i = 1, 2, \dots, p-1$ define $c(i)$ so that r_{i+1} is the $c(i)$ -th child of r_i from the left.

Child trees in $O_D - O_D^1$

If T is the parent tree of some tree, then all the children of ℓ_q are leaves. Thus if ℓ_q has a non-leaf child, then T has no child tree. Assume otherwise. Now all the children of ℓ_q are leaves, and in this case T has one or more child trees, as follows.

Let $T[i, j]$ be the ordered tree derived from T by swapping (1) the subtree rooted at ℓ_q and (2) the subtree rooted at j -th child of r_i .

By the definition of the parent tree in Section 3, for each i, j such that $i = 1, 2, \dots, p-1$ and $j = c(i) + 1, c(i) + 2, \dots, d(r_i)$, $T[i, j]$ is a child tree of T , and for each i, j such that $i = p$ and $j = 1, 2, \dots, d(r_p)$, $T[p, j]$ is a child tree of T . Note that for each i and j above the subtree rooted at j -th child of r_i is just a leaf. Intuitively, we swap the subtree rooted at ℓ_q only with a leaf locating to “the right” of “the right-down path”.

Based on the case analysis above, given an ordered tree T in O_D , we can find all child trees of T in O_D . We can find each child tree in $O(1)$ time on average. Then recursively applying the algorithm from the root tree T_D^r one can generate all ordered trees in O_D . Thus we have the following theorem.

Theorem 4.1 *One can generate all ordered trees in O_D in $O(|O_D|)$ time.*

5 Conclusion

In this paper we designed a simple algorithm to generate all ordered trees with specified degree sequence. The algorithm generates each tree in $O(1)$ time for each on average. Can we generate all unordered trees with specified degree sequence in $O(1)$ time for each?

References

- [G93] L. A. Goldberg, *Efficient Algorithms for Listing Combinatorial Structures*, Cambridge University Press, New York, (1993).
- [K06] D. E. Knuth, *The Art of Computer Programming, Fascicle 4, Generating All Trees*, Addison-Wesley Pub, (2006).
- [KL99] J. F. Korsh and P. LaFollette, *Towers, Beads, and Loopless Generation of Trees with Specified Degree*, Congressus Numerantium, Vol. 139, pp.157–166.
- [KL00] J. F. Korsh and P. LaFollette, *Multiset Permutations and Loopless Generation of Ordered Trees with Specified Degree Sequence*, Journal of Algorithms, Vol. 34, (2000), pp.309–336.
- [KL02] J. F. Korsh and P. LaFollette, *Loopless Generation of Trees with Specified Degrees*, The Computer Journal, Vol. 45, (2002), pp.364–372.

- [KS98] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms*, CRC Press, Boca Raton, (1998).
- [LN01] Z. Li and S. Nakano, *Efficient Generation of Plane Triangulations without Repetitions*, Proc. of ICALP 2001, LNCS 2076, (2001) 433–443.
- [M98] B. D. McKay, *Isomorph-free Exhaustive Generation*, Journal of Algorithms, Vol. 26, (1998) pp.306–324.
- [N02] S. Nakano, *Efficient Generation of Plane Trees*, Information Processing Letters, 84, (2002), pp.167–172.
- [N04] S. Nakano, *Efficient Generation of Triconnected Plane Triangulations*, Computational Geometry Theory and Applications, 27(2), (2004) 109–122.
- [NU04] S. Nakano and T. Uno, *Constant Time Generation of Trees with Specified Diameter*, Proc. of WG 2004, LNCS 3353, (2004) 33–45.
- [ZR79] S. Zaks and D. Richards, *Generating Trees and Other Combinatorial Objects Lexicographically*, SIAM Journal on Computing, Vol. 8, (1979), pp.73–81.